EECS-140/141 Introduction to Digital Logic Design
Lecture 4: Simplification in Logic Synthesis

I. REVIEW AND INTRODUCTION

I.A General Synthesis Procedure

I.A.1 Express Function as:

I.A.1.a Define variables and assign values

I.A.1.b (Optional) Express as logic function using:

I.A.1.c Express as Truth Table

- All possible input combinations
- Rows are represented by minterms and Maxterms:

I.A.2 Optional: Express T.T. as CSoP or CPoS

I.A.2.a Very straightforward using:

I.A.2.b CSoP has lower cost (vs. CPoS) if:

I.A.3 Simplify: find low-cost SoP or PoS synthesis

— Tricky because it relies on Boolean Algebra properties:

— This is the focus of this lecture.

I.A.4 Optional: convert to:

This reduces:

I.B B.A. Approach to Simplification

I.B.1 Start with Canonical Form (Either CSoP or CPoS)

I.B.2 Use Combining Property to merge terms (SoP) or factors (PoS)

I.B.3 Duplicate terms (or factors) as needed to combine more

I.B.4 Still tricky

Which terms/factors can be combined? When to duplicate?

We need:

II. SoP SIMPLIFICATION

We start with finding simple SoP form; will do PoS later...

II.A Karnaugh Maps (K-Maps)

- K-map is a graphical aid to manual simplification that "works" for relatively few (≤ 5) inputs (variables).
- K-map is just an alternative representation of:

  Grid instead of column for outputs.

- Can be used for either SoP or PoS simplification.

II.A.1 2-Variable K-Map

II.A.1.a Basics

Truth Table *K-Map*

a	b	minterm
0	0	
0	1	
1	0	
1	1	

Advantage: minterms that have common factor are "adjacent":

II.A.1.a Continued

CSoP includes all minterms corresponding to row with 1 in output (f) column.

So, look for rectangles of:

II.A.1.b Example

Truth Table *K-Map* *CSoP*

a	b	f
0	0	
0	1	
1	0	
1	1	

*Note*1: Must include all 1's in K-map.

*Note*2: May include K-map cells (with 1's) more than once to simplify:

II.A.2 3-Variable K-Map

II.A.2.a Basics

Similar idea, but must take care to:

K-Map:

Important: Edges "wrap around". Example: m_0 and m_4 are adjacent!

Again, terms that can be combined are grouped together in rectangles.

Groups are now:

II.A.2.b Example

Truth Table *K-Map*

a	*b*	*c*	*f*
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

II.A.2.c Another Example

Truth Table *K-Map*

a	*b*	*c*	*f*
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	1	0	
1	1	1	

Group of 4?

Group of 2 that includes "new" 1's?

II.A.3 4-Variable K-Map

II.A.3.a Basics

Keep expanding! Careful with ordering!

Now can have groups of 2, 4, or 8!

How many groups of 8?

How many groups of 4?

How many groups of 2?

Example (students verify this):

$m_0 + m_1 + m_3 + m_2 + m_8 + m_9 + m_{11} + m_{10} =$

II.A.3.b Example

II.A.3.c Another Example

II.A.4 5-Variable K-Map

- Stack 2 4-variable K-maps on top of each other
- See book.

II.A.5 More Examples of K-Maps

See Section 4.1 of book.

II.B Formal Simplification Strategy

— K-map examples showed some basic principles useful for simplification.

— More difficult cases require a more formalized strategy.

II.B.1 Terminology

— Need standardized terms to describe strategy.

— Introduce here for SoP form (PoS form later).

II.B.1.a Literal

Definition: A single input variable in either:

Examples:

II.B.1.b Implicant

Definition: A product term that includes *only* $f = 1$ minterms.

Example 1:

Example 2:

Example 3:

Implicants: 5 minterms plus:

II.B.1.c Prime Implicant (PI)

Definition: An implicant that is not *entirely* included in another implicant with:

So, you would never use a non-prime implicant if you are trying to find a simple SoP synthesis.

Example: 3-variable K-map from Example 3 above:

Prime implicants:

II.B.1.d Cover (Noun)

Definition: A *collection* of implicants that includes *all* $f = 1$ minterms (and *no* $f = 0$ minterms).

A cover corresponds to a particular SoP synthesis.

Examples:

II.B.1.e Essential Prime Implicant (EPI)

Definition: a prime implicant (PI) that includes at least one f=1 minterm that is *not* included in *any* other PI.

EPI's *will* be included in any reasonably simplified cover.

Example: 3-variable K-map from Example 3 above:

Which PI's are *essential*?

Another Example:

PI's:

EPI's:

II.B.2 Procedure

a. Identify *all* PI's.
b. Identify *all* EPI's. If these form a cover:
c. If not, select other PI's to complete the cover. There is no *one* best method for this. *One* method when faced with choice of PI to add:
   - Choose PI with fewest literals that provides additional cover.
   - If several to choose from, choose the one that:

Note: This procedure is *not* guaranteed to result in a *minimum*-cost SoP synthesis (example later).

II.B.3 Examples

a.

b.

c.

Examples continued.

d.

e. More in book!

II.B.4 Don't Care Minterms

II.B.4.a Example

Consider the following 2-person "game" with players W and X.

A bag has 3 pieces of paper labeled 1, 2, and 3.

Player W draws a piece of paper, records the number, then replaces the paper in the bag.

Player X does the same.

Player with larger number "wins".

Design a logic circuit whose output is 1 whenever player X wins or ties.

Let W_1W_0 be a binary representation of the number drawn by W. Let X_1X_0 be the same for X.

For both cases, represent:

Note: Not possible to have:

So, we *don't care* what the output is for either of those input combinations.

K-map:

Note: can indicate don't-care minterms with $D(\cdot)$.

Here:

$f(W_1, W_0, X_1, X_0) =$

II.B.4.b What to do with don't-cares?

Since they can be *either* 0 or 1, we can *choose* their value to help simplify the implementation.

We will want to make $d = 1$ if and only if that helps us get a PI with fewer literals.

Game Example:

PI's:

EPI's:

min-cost SOP synthesis:

Note: with this implementation:

III. Product of Sums (PoS) Simplification

III.A Intro

— Very similar to SoP, except we focus on:

— Min-cost PoS may be less or more costly compared to min-cost SoP.

III.A.1 Terminology

We will have PoS versions of the terms introduced earlier for SoP simplification:

PoS Implicant:

PoS Prime Implicant:

PoS Essential Prime Implicant:

PoS Cover:

III.B Examples

III.B.1 Example 3 From II.B.1.b

III.B.2 From II.B.3.c

IV. Multiple-Output Circuits

IV.A Introduction

— When a logic circuit/function has multiple outputs (with the same set of inputs), we may be able to share some portions (e.g., SoP product terms) between the output expressions. That is, if f_1 and f_2 are functions of the same inputs, it is possible to have:

— This sharing possibility may dictate the choice of PIs for one or both outputs from their individual min-cost syntheses.

— Sometimes it can even be better to use a sub-optimal synthesis for one or more outputs to obtain a min-cost multi-output circuit.

— We will focus here on SoP synthesis.

IV.B Examples

IV.B.1 f_1 from II.A.2.c and f_2 from II.B.3.a

Options:

a. Implement each separately (no sharing):

b. Share $\bar{b} \cdot \bar{c}$ term:

c. Maximize sharing from K-maps: Implement f_1 as:

Resulting logic circuit:

IV.B.2 f_1 from II.B.3.d and f_2 Below

Options:

a. f_1 with horizontals and f_2 as above (no sharing possible):

b. f_1 with horizontals and f_2 =

c. f_1 with verticals and f_2 as above:

V. Multi-Level Synthesis

V.A Introduction

V.A.1 Emphasized 2-Level Synthesis So Far

V.A.2 Reasons to Consider Multi-Level Synthesis

V.A.2.a Limits on Number of Inputs to Gates (Fan-In Limit)

Example: If f has 5 terms in SoP synthesis, 2-level circuit requires a 5-input OR. What if only 3-input OR gates are available? We *could* implement as:

Now we have a 3-level circuit since some signals travel through 3 gates to output.

But, we can often manipulate a logic expression to reduce fan-in requirement *and:*

V.A.2.b Wiring Complexity

Multi-level circuits typically require less interconnection wiring than 2-level circuits, which reduces chip or board area required.

V.A.3 A Disadvantage of Multi-Level Synthesis: Longer Propagation Delay

When inputs to a gate change, there is a delay before the output changes: *gate propagation delay.* Multi-level circuits require signals to pass through more than 2 gates input to output, thus:

V.B Factoring

V.B.1 Introduction

Starting from a min-cost SoP synthesis, it is often possible to factor out common portions of several product terms using the Distributive Property, resulting in:

V.B.2 Examples

a.

b.

V.C Functional Decomposition

V.C.1 Introduction

— Start with factoring.

— Find sub-functions that can be re-used. One simple re-use: a sub-function and:

V.C.2 Example

V.D Analysis of Multi-Level Circuits

V.D.1 Introduction

— Sometimes we need to derive a T.T. from a *given* multi-level circuit.

— This is pretty easy if you:
a) label internal points.
b) write a logic expression for each.
c) write f in terms of internal points.
d) expand back out to get:

V.D.2 Example Above